NPS-54-82-002

NAVAL POSTGRADUATE SCHOOL

Monterey, California

# SOFTWARE MAINTENANCE: IMPROVEMENT THROUGH BETTER DEVELOPMENT STANDARDS AND DOCUMENTATION

Norman F. Schneidewind

February 1982

Final Report: 1 Jan 80 to 1 Jan 82

Approved for public release; distribution unlimited.

Prepared for:
The Trident Command and Control Systems
Maintenance Agency
Newport, Rhode Island

NAVAL POSTGRADUATE SCHOOL
Monterey, California

The work reported herein was supported by the Trident Command and Control Systems Maintenance Agency.

This report was prepared by:

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>NPS-54-82-002 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>SOFTWARE MAINTENANCE: IMPROVEMENT THROUGH BETTER DEVELOPMENT STANDARDS AND DOCUMENTATION | | 5. TYPE OF REPORT & PERIOD COVERED<br>Final Report<br>1 Jan 80 to 1 Jan 82<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Norman F. Schneidewind | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Naval Postgraduate School<br>Monterey, California 93940 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>N4216680WR00007 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Trident Command and Control Systems<br>Maintenance Agency<br>Newport, Rhode Island | | 12. REPORT DATE<br>22 February 1982<br>13. NUMBER OF PAGES<br>41 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

| | |
|---|---|
| Software maintenance | Military Standard MIL-STD 1679 |
| Software maintainability | Weapons Specification WS 8506 |
| Software standards | Traceability |

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Software maintenance is frequently the most expensive phase of the software life cycle. It is also the phase which has received insufficient attention by management and software developers. Software standards have improved the ability of the software community to develop and design software. Unfortunately, most standards do not deal with the maintenance phase in a substantive way. Since maintainability has to be designed into the software and cannot be achieved after the software is delivered, it is necessary to have software standards which explicitly incorporate requirements for maintainability. Accordingly, this report suggests design criteria for achieving maintainability and evaluates Weapons Specification WS 8506 and MIL-STD 1679 against these criteria. Using these documents as typical examples of military software standards, recommendations are made for improving the maintainability aspects of software standards.

## TABLE OF CONTENTS

## BACKGROUND

This report addresses the maintenance phase of the software life cycle and the methodologies and procedures which, if employed during the software development and design phases, will improve software maintainability. We define maintenance as that activity which is concerned with making changes to software for the purpose of improving or correcting the software. Maintainability is a property of the software which makes the maintenance activity easy to perform, i.e., changes to the software are easy to incorporate and do not lead to new errors in the software.

The Trident Command and Control Systems Maintenance Agency (CCSMA) tasked the Naval Postgraduate School to study and evaluate various Navy software standards with regard to their applicability to software maintenance. The question posed was: Could these standards, accompanied by basic program documentation, such as a listing, provide adequate guidance for a new programmer to maintain software, such as that found in the Trident Command and Control Subsystem? Related to this question is the critical relationship between development and maintenance. For the most part, the standards and documentation techniques which were reviewed (e.g., MIL-STD 1679) were designed to be used for software development and not for maintenance, specifically. This situation illustrates the fundamental problem in software maintenance: inadequate attention to this important and high cost phase of the software life cycle. Since the conditions for future maintenance problems are created during development and design, the approach taken in this study was to look to these phases of the life cycle as the areas having the greatest potential for improving the maintainability of software. This concept suggests that maintainability must be designed into the software and that development methodology has a more significant effect on maintainability than maintenance practices themselves. However, pre-maintenance phase activities should not be emphasized to the exclusion of maintenance phase activities. Maintenance practices, such as patching programs and deficiencies in testing subsequent to a change (e.g., lack of regression testing), obviously exert a deleterious effect on the quality of software. Unfortunately improvements in maintenance practices cannot cure underlying problems which are introduced into the software when it is specified and designed. Hence it is clear that significant gains in maintainability can only be achieved by recognizing maintainability objectives as an integral part of development and design.

Consistent with the above problem approach, this report begins with a discussion of the findings of a number of researchers, in the areas of software development and maintenance practices, which bear on the problem of improving software maintainability. This is followed by an evaluation of Navy weapon system software standards (WS 8506 and MIL-STD 1679), from the standpoint of their effectiveness as aids to the maintenance programmer.

# I. INTRODUCTION

Software maintenance is one of the most expensive phases of the system life cycle. One author suggests it is as high as 67 per cent of the effort in large-scale systems [1]. Despite this fact, maintenance requirements receive little attention during system development. There are many reasons for this situation, but three stand out:

1. Maintenance is considered less glamorous, interesting and challenging as compared to system design and programming; hence, there is little incentive for computer personnel to become involved in maintenance activities.

2. During development it is often too early in the project to foresee problems which may occur during maintenance; as a consequence maintainability is not provided in the system design.

3. Project management does not always recognize that maintainability considerations should be an integral part of the design process.

One result of this lack of attention to maintainability requirements during design is loss of traceability. This is defined as the ability to identify the technical information which pertains to a software error which has been detected during the maintenance phase and thereby trace the error to the applicable design specifications and user requirements statements. The same traceability capability is also required for software improvements which are made during the maintenance phase. It is clear that if significant improvements are to be made in software maintainability, requirements for maintenance must be made an integral part of the software specification development and design process. In fact, as mentioned later, some software developers recommend that software design be oriented around the need to maintain the software in its operating environment.

## II. OBJECTIVES

The objectives of this report are:

1. In particular, to evaluate Weapons Specification (WS) 8506 [2] and Military Standard (MIL-STD) 1679 [3] with regard to their suitability for computer program maintenance in embedded computer systems.

2. In general, to discuss and propose software development practices which will lead to improved maintainability for any software system.

In 1. emphasis will be on the evaluation of the ability of these documents to provide for traceability during program maintenance.

## III. APPROACH

The WS 8506 and MIL-STD 1679 documents were examined with regard to their effectiveness for software maintenance purposes--in particular, their capability for providing traceability. In addition to this specific aspect, all sections of these documents which are applicable to maintenance were reviewed, and strong and weak points were noted. A major approach in this review was the use of a rich body of literature in software engineering which was employed as an information source for suggesting improvements in the documents. This literature and our concept of the characteristics of good specifications and standards for maintenance were used to develop a set of criteria for judging WS 8506 and MIL-STD 1679.

# IV. GOOD MAINTENANCE PRACTICES

## A. Design Approaches for Achieving Good Maintainability

As described in Yourdon [4], a good software design and maintainability strategy is that of divide and conquer. The cost of implementation will be minimized when parts of the problem are manageably small and separately solvable. Similarly, the cost of maintenance will be minimized when parts of the system are easily related to the application (a facet of traceabilty), manageably small and separately correctable. <u>This modularity concept should be incorporated in weapons systems software specifications and standards</u>. Furthermore, pieces of the problem should be independent. Highly interrelated parts of the problem should be in the same piece of the system and unrelated parts should reside in unrelated pieces of the system. Finally, implementation and maintenance will be minimized when each relationship between pieces of the system corresponds only to a relationship between pieces of the problem.

Of great interest for maintenance is the fact that it is very difficult to simplify the structure of a program after the program has been written. If modularization is undertaken at this point, new interfaces must be designed, thus possibly increasing the program's complexity. Also the entire program must be checked to determine which parts should be changed and which parts should remain unchanged, as a result of modularization. Thus once reduced to code, the structural complexity of a program is essentially fixed; therefore, it is impractical to simplify software during the maintenance phase.

Peters [5] writes that much of the difficulty of software design stems from the fact that the problem we are attempting to solve is changing while we are solving it. Some of these changes are made by the designer as he obtains a better understanding of the problem, and other changes come from the user. In either case destabilization of the project and reduction in quality are the results. <u>This situation illustrates the need for software specifications to require a formal change procedure</u>. Also, Peters warns agains the fanaticism of some promoters of design methods and techniques, pointing out that each method is directed at an idealized problem, not necessarily the one to be solved. Although there may be some merit to this argument, the use of structured design and programming techniques, albeit imperfect, would still achieve consistency of documentation and a disciplined approach to design and programming.

Lientz [6] has mentioned that a principle of good design is to design with enhancements in mind because, in his survey, it was found that most (48 per cent) of maintenance activities stemmed from enhancements. This, of course, has a direct bearing on maintainability. This is a point well made, but one that is difficult to implement in practice due to the difficulty of anticipating future operational requirements during the design stage. Since many enhancements are the result of changing data requirements, it is possible, however, to avoid certain practices which are detrimental to maintenance, such as imbedding data descriptions in programs. <u>Hence, a software specification and standard should call for independence among program code, data and data bases</u>.

Heninger and colleagues at the Naval Research Laboratory have developed, in conjunction with the A7E aircraft computer system, the concept of changes which are likely to occur and those which are not likely to occur [7]. A related matter is the identification of functions which maintainers would like to modify or remove easily, should the need arise. Easy changes should correspond to the most likely changes. These considerations evolve into a design principle of separating things that will remain the same, no matter what changes are made in the rest of the system, from those things that will be affected by the changes. System documentation should only specify external behavior without implying a particular implementation. For example, programs should not have to be changed significantly if changes are made to the hardware. In other words, software is described without reference to hardware. Another example of the desired immutability of software is that software should not change if data arrives in different formats over different channels. Thus from the above, <u>a software specification and standard should specify that software functions be divided into those which will (likely) change and those which will not (unlikely) change</u>. This approach will have the desirable result of causing the designers and programmers to think about potential change requests early in system development.

## B. Specification and Documentation Requirements for Achieving Good Maintainability

One of the best approaches for designing a system is to design the documentation first. Considerable emphasis should be placed on documentation for maintaining software. Hegland suggests determining documentation requirements at the beginning of a project [8]. Young [9] lists the following types of documentation as being applicable to computer software:

| Type | Project Phase Produced In |
|---|---|
| Functional Requirements Document | Problem Definition |
| Data Requirements Document | Problem Definition |
| System and Sub-System Specifications | System Design |
| Program Specification | System Design |
| Data Base Specification | System Design |
| Test Plan | System Design |
| User Manual | Programming |
| Operations Manual | Programming |
| Program Maintenance Manual | Programming |
| Test Analysis Report | Test |

To the above should be added an Interfaces Specification, which would be produced during system design. <u>A software specification and standard should require that the documentation to be provided on a project be specified</u>. It should also be required that the <u>various levels of documentation be consistent</u> (e.g., sub-program specifications should be consistent with the associated program specification).

Since good maintainability implies good readability [10], the following aids to readability should be required by the specification and standard:

- Comments per line or related lines of code.
- References in the source listings to software specifications, and approved changes and test plans.
- References in all documents to related documents.

Jones [11], based on his study of design and specification techniques, concluded it is impossible to conduct a large software project without considerable oral communication among project members; it is not possible, in his view, to conduct a project mainly on the basis of written documentation. One reason for this situation is that written documentation may become so bulky that it ceases to be useful. As a consequence, according to Jones, word processing costs have become the second largest project expense, behind debugging costs. Despite this result, we would not propose a reduction in project documentation; on the contrary, the thoroughness of documentation should increase in quality and quantity. However, the possibility of inundating a project with paperwork does suggest the need for a specification and standard to stipulate oral communication in the form of design reviews and walk-throughs.

Balzer and Goldman [12] mention understandability as the first criterion for judging specifications. Since the specifications are the basis of a contract between contractor and customer, they must be clear and unambiguous. It seems appropriate, therefore, for a software standard to state that a primary objective of the project specifications is to achieve understandability for both the contractor (developer) and customer (user and maintainer). These authors also make the interesting point that the specification itself must be testable. That is, for a specification to be considered valid, it is necessary to demonstrate that questions (tests) that one might pose concerning proposed system operations can be answered satisfactorily by the specifications. Thus, a software standard should require that the exercise of "testing" the specification for validity be a part of the specification development process. This consideration suggests the need for a specification model. A carefully designed standard could serve as the model for writing a specification. Furthermore,the authors state the importance of making the specification independent of the implementation. Thus a specification should state what is to be accomplished and not how it is to be implemented. Consequently, another objective of specification development which should be stated in a software standard is the objective of achieving independence of the specifications from methods of implementation. The current interest in specification languages as a means of obtaining consistency and understandability fo specifications has implications for maintainability. Balzer and Goldman caution against the use of a specification language which optimizes (reduces the resources required to execute a program) a system. This would have the undesirable effects of reducing understandability, testability and maintainability. Thus a software standard should stipulate that optimization techniques are not to be employed unless their use is unavoidable due to performance requirements. The reason for this is that tricky coding techniques, designed to achieve optimization, frequently lead to unmaintainable software.

Lastly, these authors discuss the effects of specification changes on maintainability. They recommend that a specification be so designed that changes can be localized and not have an effect on the rest of the specification. However, there are instances in which different specifications, such as system and program specifications, or different parts of a given specification, must be related. Where these relationships exist, this fact must be visible in the specification so that changes to related specifications can be made when a change is made in a given specification. Thus, with regard to the above two points, <u>a software standard should stipulate that parts of a specification are to be made as independent as possible; but, where relationships must exist, these relationships must be made explicit in the system documentation</u>.

One of the problems with system development is lack of adherence to stated software performance objectives over the software's life cycle. As stated by McCall [13], little attention is given to identifying the qualitiies that the software should exemplify over its life cycle. What is needed is a clear statement of performance goals in the user requirements statement, consistency in the use of these goals in subsequent stages of development and the ability to trace these goals forward, from user requirements phase to maintenance phase; and backward, from maintenance to user requirements. The achievement of these objectives will be aided if a <u>software specification and standard requires that project documentation identify key performance requirements and state how the implementation is to satisfy performance requirements</u>.

## C. Testing Approaches for Achieving Good Maintainability

DeMillo and colleagues [14] believe a formal demonstration (test) that a program is consistent with its specifications has value only if the specifications are derived independently from the program. In other words, the specifications should be derived from user requirements and not from the program. Then, if the specification reflects user requirements and the program is in accordance with the specification, a demonstration of the program would be meaningful. If the above is not the case, the following situation could ensue: A program fails, it is changed, and the changes are based on faulty specifications; or the specifications are changed, and those changes are based on knowledge of the program gained through the failure. In either case, the requirement of using independent criteria is no longer met. To guard against this situation, it is necessary for a software standard to specify that software design or performance specifications be independently derived from user requirements and not, after-the-fact, from the program design. Also, as pointed out by Ramamoorthy and colleagues [15] the test plan should be independent of the design specification but dependent upon user requirements. If this is not the case, an error in translation from the user requirements to the specification will not be caught if the test plan mirrors the specification. The error will be caught if the test plan reflects user requirements. In other words, software must be tested against what the software was intended to do and not against what it is doing.

A big step towards error reduction in software would result from the use of standard and certified reuseable modules [11]. This practice would obviate the need for module testing of reused modules. Only integration testing would be necessary for these modules. The practicality of this approach is less apparent in embedded computer applications than in the commercial applications due to the nonstandard nature of the former (e.g., uniqueness of some electronic warfare programs relative to accounts payable routines). However, certain data transformation and algorithmic routines may be standard (more or less). A Software standard should require the use of reuseable modules wherever appropriate with standard interfaces between modules.

Design problems are the chief source of programming errors and detecting and removing errors are the major programming costs [11]. Therefore, it is of interest to identify the defect removal techniques which work best with a given design approach. Jones [11] lists four defect removal methods:

- Design reviews.
- Testing.
- Correctness proofs.
- Models or prototypes for verifying designs.

Two major categories of design error are: (1) leaving out needed user functions, and (2) putting in functions that users don't need. Correctness proofs and testing are not guaranteed to find these errors. However, methods for depicting models of systems, such as Hierarchy Plus Input-Process-Output (HIPO) charts [16] and Composite Design [17], in combination with design reviews, do provide considerable help in identifying missing, unneeded, redundant and ambiguous functions and in specifying how the functions should be related and coupled together. These methods have an important ancillary benefit by providing a uniform method of system documentation. In view of the above, it is important to not rely on one or two methods, such as correctness proofs, to provide product quality assurance, but rather to employ a variety of methods where the choice of method will depend upon the objective of the test. Thus, a software specification and standard should not be restrictive in their stipulation of product quality assurance and testing methodologies. In fact, it would be worthwhile for these documents to enumerate the various methods. (At this time, methods for verifying the design against user requirements coupled with formal design reviews are the best methods for assuring product quality.) In addition, consideration should be given to the adoption of a documentation technique, such as HIPO, as a standard for embedded computer development. As noted by Pariseau [18] something like HIPO would have improved visibility and documentation on the Carrier Based Tactical Support Center project. In addition to the use of a documentation standard, one of the most important improvements that could be made in the documentation specification is the use of examples, perhaps ones from past projects. Wtihout examples, specifications and standards appear ethereal to the designer and programmer. A requirement for a software specification and standard to show examples would significantly improve their usability. A related consideration is that the choice of documentation technique should not be left to the contractor (developer) alone to decide.

Schneidewind [19] states that stress or saturation testing is as applicable to software as to hardware, particularly with regard to pre-mission testing, when it is essential to identify marginal hardware and software (e.g., software which operates satisfactorily under normal load but fails--buffer overrun--under peak load) prior to the mission. One cannot assume that the software remains invariant between missions, because program changes could result from hardware errors (e.g., a transient hardware error causes changes in programs residing in memory or on disks), program maintenance activities or errors in revised programs which are distributed to the field.

Other test procedures which will improve maintainability are:

- Static testing (without execution).
- Dynamic testing (with execution).
- Regression testing (retest of all modules affected by a software change).
- Testing specifically for the response of the system to undesirable or unexpected events. This includes stress testing.
- Providing for the repeatability of tests.
- Use of an independent quality control group.

Thus <u>a software specification and standard should make specific reference to the above techniques</u>.

# V. EVALUATION OF WEAPONS SPECIFICATION WS 8506

The Fiscal Year 1979 Command and Control System Maintenance Agency (CCSMA) System-Level Software Maintenance Approach and Transition Plan [20] states that WS 8506 and MIL-STD 1659 will be among the predominant factors for software delivered to CCSMA for the Trident ships. Furthermore, CCSMA is determining what software logic documentation is necessary for maintenance and where the voids are. The transition plan from the development to maintenance phase includes:

- Test and evaluation.
- Validation of documentation.
- Configuration management.
- Introduction of system into the fleet.
- Software maintenance.
- Quality assurance and audits.

The software delivered to CCSMA by the development agency will include:

- Computer program performance specifications.
- Computer program design specifications.
- Interface design specifications.
- Computer program operator's manual.
- Test specification requirements.
- Test procedures.

Thus it is of interest to evaluate WS 8506 in this section and MIL-STD 1679 in the next section, particularly with regard to their suitability for maintenance. This evaluation is performed by applying the underlined criteria developed in Section IV. Only the criteria which are applicable to WS 8506 are utilized in this section. The evaluation of both WS 8506 and MIL-STD 1679 will categorize the two documents as follows:

- No coverage.
- Inadquate coverage.
- Adequate coverage.
- Excellent coverage.

In addition to the evaluation against the criteria, comments will be made about various sections of the documents, where appropriate.

TABLE 1
WS 8506 EVALUATION

| Criterion | WS 8506 Status |
|---|---|
| A. Design Approaches. | |
| 1. Modularity. | Adequate coverage in Sec. 6.2, Computer Program Design Specification Detailed Requirements. |
| 2. Change procedure. | No coverage. |
| 3. Independence of code, data and data base. | No coverage. |
| 4. Separation of software functions by anticipated degree of change. | Adequate coverage in Sec. 5.2, Computer Porgram Performance Specification Detailed Requirements. |
| B. Specification and Documentation Requirements. | |
| 1. Specification of documentation. | Excellent coverage throughout. |
| 2. Consistent documentation. | Excellent coverage throughout. |
| 3. Readability. | No coverage. |
| 4. Design reviews and walkthroughs. | No coverage. |
| 5. Identification of performance requirements. | Adequate coverage in Sec. 5.2., Computer Program Performance Specifica- Detailed Requirements. |
| C. Test Requirements and Relationship to Specification Documentation and Design. | |
| 1. Statement of product assurance methods. | Adequate coverage in Sec. 11, Computer Program Test Plan. |
| 2. Documentation standard. | No coverage. |
| 3. Use of documentation examples. | Excellent coverage throughout. |
| 4. Regression testing. | No coverage. |

TABLE 1 (Continued)

| Criterion | WS 8506 Status |
|---|---|
| 5. Undesirable and unexpected event testing, including stress testing. | No coverage. |
| 6. Repeatability of tests. | No coverage. |
| 7. Independent quality control. | No coverage. |

It is not surprising that WS 8506 does not mention the "No coverage" items in TABLE 1 since these items pertain to technical and management advances in software which had not occurred when WS 8506 was issued in November 1971. From a maintenance standpoint the most serious voids are lack of change procedure; documentation standard, such as HIPO; and regression testing. The lack of these capabilities impairs traceability. The use of Data Item Descriptions (DID's) to cover these voids might be considered if WS 8506 is to continue in use for embedded computer systems. With regard to Item A.1, modularity, there is coverage with respect to allocating software modules by function, e.g., tracking, but no discussion of structured programming since this technology was not available at the time of publication.

## Other Comments

Comments are made below about various sections of WS 8506 independent of the criteria for evaluation which were used in the foregoing.

1. Although computer performance specification is a commonly used term in military software, it could be inappropriate on two counts: (1) functional specification would seem more appropriate to connote the idea of implementing user functional requirements, and (2) the word "performance" is frequently used to denote performance measurement (CPU and memory utilization).

2. P. 3-2, paragraph 3.13. The use of the word "operation" seems inappropriate. Better terminology might be task or process (in operating system sense) to denote activity on a program which, in turn, is implemented with computer instructions.

3. P. 4-1. Other types of documentation, such as decision tables, data flow diagrams and directed graphs, could be mentioned.

4. P. 5-1, paragraph 1.2. A matrix would be useful for depicting function dependencies.

5. P. 5-2, section 3. The distinction among functional, operational and performance requirements is not clear; some seem to be subsets of the others.

6. P. 5-3. The various types of documentation should be related in a heirarchy, possibly with HIPO charts.

7. P. 5-7. The use of state diagrams could be useful for showing processing states and state transitions.

Paragraph 3.3.N.2. Processing should be related to the use of data bases.

8. P. 5-10, paragraph 3.4. Discussion of data base; system growth; recovery; and capacity, storage and time requirements is good.

9. P. 5-11, paragraph 4.1. All tests should be documented.

Paragraph 4.2. Discussion of use of tools and tolerances is good.

Paragraph 4.3. Discussion of acceptance testing and criteria is good. Should specify which items can be tested by simulation.

10. P. 6-1. Technique for specifying programs is good.

Section 3.1. Allocation of functions to programs is good.

11. P. 6-2. Identification of tasks is good.

Paragraph 3.2. Functional description is good.

Paragraph 3.3. Discussion of allocation of storage and processing time, sequencing requirements and equipment constraints is good. Figure 6-2 is helpful.

Paragraph 3.4. Mention of flow, both control and data is particularly good.

12. P. 6-7. paragraph 3.4.1. Discussion of interrupts is good.

Paragraph 3.4.2. Discussion of subprogram reference is good.

Paragraph 3.5. Discussion of use of monitor, loader, etc. is good; even talks about configuration management. Mention of labeling conventions is good.

13. P. 6-8, section 4. This Quality Assurance section should be expanded to include discussion of independent quality control group and organization of the quality control function.

14. P. 7-1. Considering system subroutines in the same light as subprograms is good. The question arises as to the appropriateness of program level documentation for operational personnel; the programming language orientation seems inappropriate. WS 8506 should be updated to reflect the use of stacks, reentrant code, separation of code and data and use of concurrent processes.

15. P. 7-3, paragraph 3.3. Discussion of data base usage is good.

16. P. 7-5, paragraph 3.3.1. Discussion of table documentation is good. The discussion of bit layouts and flags is incompatible with today's emphasis on use of higher order languages (HOL). This section should be revised to reflect greater HOL usage. Perhaps the specification could be divided into different parts depending upon whether assembly language or HOL is used.

17. P. 7-6, paragraph 3.4. Discussion of I/O formats is good; disc formats should be added.

18. P. 7-8, paragraph 3.5. Description of system subroutine naming and referencing is good. Might consider use of decision tables for stating conditions of subroutine use.

19. P. 7-9. Diagram of subprogram relationships is good.

20. P. 8-1, section 8.1. Good-tie back to other parts of specification.

Section 8.2. Mention of options good.

21. P. 8-2, paragraph 3.1. Mention of table indexing and initial condition procedures is good.

22. A list of acronyms would be helpful.

## Summary

WS 8506 is considered to be well thought out, comprehensive and a very good specification for the documentation of program development, particularly in view of the early publication date of this document. The strategy of making each level of documentation responsive to the next upper level (subprogram design under program design), represents foresight in the use of top-down design prior to the time this term was in vogue. It cannot be faulted for not including programming technologies which had not been developed at the time of its publication. However, the document is less useful for maintenance purposes for the reasons previously given.

# VI. EVALUATION OF MILITARY STANDARD MIL-STD 1679

MIL-STD 1679 is of great importance today because it is being considered for recommendation for adoption as a Department of Defense software standard, along with MIL-S-52779 as a Department of Defense software specification, by the Joint Logistics Commanders, Joint Policy Coordinating Group on Computer Resource Management [21].

The evaluation of MIL-STD 1679 against the applicable criteria developed in Section IV follows.

TABLE 2
MIL-STD 1679 EVALUATION

| Criterion | MIL-STD 1679 Status |
|---|---|
| A. Design Approaches | |
| 1. Modularity. | Excellent coverage in Sec. 5.2, Program Design Requirements. |
| 2. Change Procedure. | Excellent coverage in Sec. 5.11.2, Configuration Control. |
| 3. Independence of code, data and data base. | Adequate coverage in Sec. 5.4.1, Symbolic Parameterization. |
| 4. Separation of software functions by anticipated degree of change. | Inadequate coverage in Sec. 5.1.2.7, Adaptive Parameters. |
| B. Specification And Documentation Requirements | |
| 1. Specification of documentation. | Excellent coverage. Examples: Sec. 5.6.1, Supporting Information for Program Performance Requirements; Sec. 5.1.2.3., Applicable Documentation for Program Performance Requirements; and Sec. 6.1, Contract Data Requirements. |
| 2. Consistent documentation. | Adequate coverage in Sec. 4.5, Configuration Management. |
| 3. Readability. | Adequate coverage in Sec. 5.3.10, Indentation; and Sec. 5.4.4.2, Comment Statements. |
| 4. Design reviews and walk-throughs. | Excellent coverage in Sec. 5.9.1.3, Design Reviews. |
| 5. Understandability. | No coverage. |
| 6. Specification testing. | Adequate coverage in Sec. 5.12.3.3, Documentation Reviews. |
| 7. Implementation independence of specification. | No coverage. |

TABLE 2 (Continued)

| Criterion | MIL-STD 1679 Status |
|---|---|
| 8. Non-use of optimization techniques. | Adequate coverage in Sec. 4.2, Design Requirements, but inadequate coverage in Sec. 5.4.5.1, Execution Efficiency. |
| 9. Independence of specification parts. | No coverage. |
| 10. Identification of performance requirements. | Excellent coverage in Sec. 5.9.1.4, Program Design. |
| C. Test Requirements and Relationship to Specification Documentation and Design. | |
| 1. Independence of performance specifications and software design from program design. | No coverage. |
| 2. Use of reuseable modules. | No coverage. |
| 3. Statement of product assurance methods. | Excellent coverage in Sec. 5.9, Quality Assurance; and Sec. 5.10, Program Acceptance. |
| 4. Documentation standard. | Inadequate coverage. Only structured programming (Fig. 1) conventions are used. |
| 5. Use of documentation examples. | Inadequate coverage. Only one Example in Fig. 1, Control Structures. |
| 6. Regression testing. | No coverage. |
| 7. Undesirable and unexpected event testing including stress testing. | Excellent coverage in Sec. 5.10.2.6, Software Quality Test Stress Testing. |
| 8. Repeatability of tests. | No coverage. |
| 9. Independent quality control. | Excellent coverage in Sec. 5.9.1.1, Reporting Level. |

MIL-STD 1679 includes developments in programming technology and management, such as structured programming and walkthroughs which have occurred since the advent of WS 8506. The major deficiencies are that no mention is made of the need to independently derive performance requirements from user requirements and the need to achieve relative independence of the various parts of the system and programs. From a maintenance standpoint, the former capability is important in order to achieve traceability. The latter capability is important in order to localize the effects of maintenance changes. Also, the lack of a regression testing specification is a hindrance for maintenance because this type of testing should be performed subsequent to maintenance modifications.

## Other Comments

Comments are made below about various sections of MIL-STD 1679 independent of the criteria for evaluation which were used in the foregoing.

1. P. 1, section 1.2. It is good to have included firmware in the standard.

2. P. 6, section 4 2. It is good to mention that design complexity and system interdependencies should be minimized.

Section 4.3. It is good to stipulate use of a HOL.

3. Section 4.5. It is good to specify configuration management for correlating documentation with the program for maintenance purposes.

4. P. 7, section 5.1. It seems inappropriate for the contractor, alone, to determine program performance requirements.

5. P. 8, section 5.1.2.5.c. and d. Same comment as #4. with regard to intersystem interface and function description.

6. P. 9, section 5.2. Same comment as #4. with regard to program architecture.

Section 5.2.2.3a. Predicting the rate of interrupt occurrences would be very difficult.

7. P. 10, section 5.2.3. Seems to be repetitious of Section 5.1.2.5.c.

Section 5.3.1. Use of words "compile-time system" is not clear.

Section 5.3.8. Allowing compiler to generate backward jumps seems inconsistent with objective of this section.

8. P. 14, section 5.4.3. Numerical conventions should be established by the government.

9. P. 15, section 5.4.6. Although flow charts may not be a necessary part of documentation, some sort of graphic documentation, such as block diagrams or control/data flow graphs, should be required in addition to program listings.

Section 5.5.2. Same comment as #4 with regard to resource management.

Section 5.5.3. Repetitious of section 4.3.

10. P. 16, section 5.5.4. Patches should be disallowed.

Section 5.7. Same comment as #4. with regard to program operation.

11. P. 17, section 5.8.1. Should specify the test data to be used with module testing.

Section 5.8.2. Implies bottom-up testing. Standard should not preclude the use of top-down testing.

12. P. 18, section 5.8.5. Same comment as #4. with regard to software trouble reporting.

Section 5.8.5.1.b. Add the words "and the documentation is defective" to the end of sentence.

13. P. 19, section 5.8.5.1.d. This could be classified as a specification trouble.

Section 5.9.1.2. Participation in audits should start earlier with user requirements definition.

14. P. 20, section 5.9.1.6. Same comment as #4. with regard to conduct of tests.

Section 5.10. Program acceptance seems to be based on some strange criteria, such as unresolved software and documentation errors and extent of patches.

Section 5.10.2. Should specify how test time is to be determined. The use of the words "reasonably free" should be defined. It seems that if the software is stressed beyond its design capacity, the result will be catastrophic.

15. P. 21, section 5.10.2.7. It seems this section could be more appropriately named performance test.

16. P. 22, section 5.10.2.8. Perhaps what is meant here is to cut power rather than secure power.

Section 5.10.3. and section 5.10.3.2. Use of patch limits as a software quality test limit is poor.

17. P. 23, section 5.11. Configuration management should include hardware.

18. P. 24, section 5.11.2.3. Same comment as #4. with regard to Software Configuration Control Boards.

19. P. 26, section 5.12.3.3.b. It would seem more appropriate for the government to schedule documentation reviews.

## Summary

Overall, MIL-STD 1679 appears to be a good software standard for embedded computer software, primarily because it addresses areas not emphasized in previous standards (e.g., change control). It seems to be the best standard available, both for development and maintenance, although it should be noted that its companion document for DOD adoption, MIL-S-52779, is intended only for software acquisition and is encouraged but is not mandatory (as of this writing) for use on maintenance contracts. There are two disturbing aspects of MIL-STD 1679, one of which could have a serious effect on maintenance; this is the allowance of patches to the extent of the limits given in Section 5.10.3.2. The other is the seemingly excessive control and responsibility which is delegated to the contractor.

## LIST OF REFERENCES

1. Marvin V. Zelkowitz, et. al., Principles of Software Engineering, Prentice-Hall, 1979.

2. Weapons Specification WS 8506, Revision 1, Code Ident. 10001, Requirements for Digital Computer Program Documentation, Naval Ordnance Systems Command, Department of the Navy, Washington, D.C. 1 November 1971.

3. Military Standard MIL-STD 1679 (Navy) Weapon System Development, AMSC No. 23033, FSC IPSC, Department of Defense, Washington, D.C., 1 December 1978.

4. Edward Yourdon and Larry L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Sytems Design, Prentice-Hall, 1979.

5. Lawrence J. Peters, "Design Practices to Effect Software Quality," Software Quality Management, John D. Cooper and Matthew J. Fisher (eds.), PBI-Petrocelli Books, Inc, pp. 185-196, 1979.

6. B.P. Lientz, et. al., "Characteristics of Application Software Maintenance," Communications of the ACM, Vol. 21, No. 6, pp. 466-471, June 1978.

7. Kathryn L. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," Proceedings of the Conference on Specifications of Reliable Software, IEEE Computer Society, pp. 1-14, April 1979.

8. Robert R. Hegland, "Flexibility Provisions and Document Type Selection," Documentation of Computer Programs and Automated Data Systems, National Bureau of Standards Special Publication 500-15, Mitchell A. Krasny (ed.), pp. 19-21, July 1977.

9. Roy A. Young, "Life Cycle Concepts and Document Types," pp. 10-12, Reference (8).

10. D.J. Harris, "Software Development for Tornado - A Case History from the Reliability and Maintainability Aspect," AGARD, Symposium on Avionics Reliability, Its Techniques and Related Disciplines, Ankara, Turkey, April 1979.

11. Capers Jones, "A Survey of Programming Design and Specification Techniques," pp. 91-103, Reference (7).

12. Robert Balzer and Neil Goldman, "Principles of Good Software Specification and Their Implications for Specification Language," pp. 58-67, Reference (7).

13. James A. McCall, "An Introduction to Software Quality Metrics," pp. 127-142, Reference (5).

14. Richard A. DeMillo, Richard J. Lipton, and Alan J. Perlis, "Social Processes and Proofs of Theorems and Programs," Communications of the ACM, Vol. 22, No. 5, pp. 271-280, May 1979.

15. C.V. Ramamoorthy, "The Impact of Software Specifications and Design on Test and Verification Methods," Computer Science Division and the Electronics Research Laboratory, University of California, Berkeley, 23 pages.

16. J.F. Stay, "HIPO and Integrated Program Design," IBM Systems Journal, Vol. 105, No. 2, pp. 143-154, 1976.

17. Glenford J. Myers, Composite/Structured Design, Van Nostrand Reinhold Co., 1978.

18. Richard J. Pariseau, Technical Note, "Improved Software Productivity for Military Computer Systems Through Structured Programming," Report No. NADC-76044-50, 12 March 1976.

19. Norman F. Schneidewind, "The Applicability of Hardware Reliability Principles to Computer Software," pp. 171-181, Reference (5).

20. Steven W. Oxman, Fiscal Year 1979 CCSMA System-Level Software Maintenance Approach and Transition Plan, (Revision E), Software Systems Department, Trident Command and Control System Maintenance Agency, Newport, R.I., March 20, 1979.

21. Report of the Panel on Standards for Software Quality, Joint Logistics Commanders, Joint Policy Coordinating Group on Computer Resource Management, Software Workshop, Monterey, CA, April 1979.

## DISCLAIMER

## DISTRIBUTION LIST

| | No. Copies |
|---|---|
| Prof. Victor Basili<br>Department of Computer Science<br>University of Maryland<br>College Park, MD 20742 | 1 |
| Mr. Laszlo Belady<br>IBM Corporation (VAL)<br>Old Orchard Road<br>Armonk, NY 10504 | 1 |
| Dr. Barry Boehm<br>Software Research and Technology<br>Defense and Space Systems Group<br>TRW<br>One Space Park<br>Redondo Beach, CA 90278 | 1 |
| Dr. Ned Chapin<br>Info Sci, Inc.<br>Box 7117<br>Menlo Park, CA 94025 | 1 |
| Mr. John Cooper<br>Anchor Software<br>4111 Century Court<br>Alexandria, VA 22312 | 1 |
| Prof. Lyle Cox<br>Code 52Cl<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93940 | 1 |
| Mr. Barry Deroze<br>TRW/DSSG<br>Mail Station R2 1410<br>One Space Park<br>Redondo Beach, CA 90278 | 1 |
| Mr. William Ferrara<br>Trident CCSMA<br>Building 12<br>U.S. Navy<br>Newport, RI 02840 | 1 |

| | |
|---|---|
| Dr. Matthew Fisher<br>No. 1 Violante Court<br>Eatontown, NJ 07724 | 1 |
| Mr. Jan Fränlund<br>TELUB AB<br>Ljungadalsgatan 2, Box 1232<br>SE-351 12<br>Växjö, Sweden | 1 |
| Mr. Tom Gilb<br>Infotect<br>Iver Holters Vei 2<br>N-1410<br>Kolbotn, Normway | 1 |
| Mr. Gerald Goulet<br>Naval Air Development Center<br>Warminister, PA 18974 | 1 |
| Dr. Robert Grafton<br>Code 437<br>Office of Naval Research<br>800 N. Quincy St.<br>Arlington, VA 22217 | 1 |
| Ms. Kathryn Heninger<br>Naval Research Laboratory<br>Information Processing Systems Branch<br>Communications Science Division<br>Washington, DC 20375 | 1 |
| Prof. Melvin Kline<br>Code 54Kx<br>Administrative Sciences Department<br>Naval Postgraduate School<br>Monterey, CA 93940 | 1 |
| Mr. Robert G. Lanergan<br>Raytheon Company<br>Missile Systems Division<br>Hartwell Road<br>Mail Stop BLA 1-4<br>Bedford, MA 01730 | 1 |
| Prof. Bennet Lientz<br>Graduate School of Management<br>University of California<br>Los Angeles, CA 90024 | 1 |

| | |
|---|---|
| LTC Casper H. Lucas<br>HQ, AFLCILOEC<br>Wright-Patterson AFB, OH 45431 | 1 |
| Prof. Norman Lyons<br>Code 54Lb<br>Administrative Sciences Department<br>Naval Postgraduate School<br>Monterey, CA 93940 | 1 |
| Mr. Paul A. Mauro<br>Hughes Aircraft Company<br>Bldg. 618, Mail Stop B218<br>P.O. Box 3310<br>Fullerton, CA 92634 | 1 |
| Mr. Jim McCall<br>GE Space Division<br>1277 Orleans Drive<br>Sunnyvale, CA 94086 | 1 |
| Dr. Edward Miller<br>Software Research Associates<br>P.O. Box 2432<br>San Francisco, CA 94126 | 1 |
| LCDR Ronald W. Modes<br>Code 52Mf<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93940 | 1 |
| Mr. John B. Munson<br>Corporate Software Engineering<br>System Development Corp.<br>2500 Colorado Ave.<br>Santa Monica, CA 90406 | 1 |
| Mr. John Musa<br>Bell Telephone Laboratories<br>Rm. 3A332<br>Whippany Rd.<br>Whippany, NJ 07981 | 1 |
| Dr. Peter Neumann<br>SRI International, EL 301<br>Menlo Park, CA 94025 | 1 |

Mr. Steve Oxman 1
Trident CCSMA
Building 132T
U.S. navy
Newport RI 02840

Mr. Richard Pariseau 1
Naval Air Development Center
Warminister, PA 18974

Dr. David Parnas 1
Information Processing Systems Branch
Communications Science Division
Naval Research Laboratory
Washington, DC 20375

Prof. C.F. Ramamoorthy 1
Department of Electrical Engineering
and Computer Science
University of California
Berkeley, CA 94720

Prof. Norman F. Schneidewind 10
Code 54Ss
Administrative Sciences Department
Naval Postgraduate School
Monterey, CA 93940

Dr. John Stancil 1
Trident CCSMA
Building 132T
U.S. Navy
Newport, RI 02840

Dr. Leon Stuki 1
Boeing Computer Services
Seattle, WN 98124

Mr. David M. Weiss 1
Information Processing System Branch
Communications Science Division
Naval Research Laboratory
Washington, DC 20375

Prof. Roger Weissinger-Baylon 1
Code 54Wr
Administrative Sciences Department
Naval Postgraduate School
Monterey, CA 93940

|  |  |
|---|---|
| Prof. Steven Yau<br>Department of Computer Science<br>Northwestern University<br>2001 Sheridan Rd.<br>Evanston, IL 60201 | 1 |
| Prof. Marvin Zelkowitz<br>Department of Computer Science<br>University of Maryland<br>College Park, MD 20742 | 1 |
| AS/OR Library<br>Code 54/55<br>Naval Postgraduate School<br>Monterey, CA 93940 | 1 |
| Computer Center Library<br>Code 0141<br>Naval Postgraduate School<br>Monterey, CA 93940 | 2 |
| Computer Science Department<br>Code 52<br>Naval Postgraduate School<br>Monterey, CA 93940 | 2 |
| Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 23314 | 2 |
| Knox Library<br>Code 0142<br>Naval Postgraduate School<br>Monterey, CA 93940 | 4 |
| Office of Research Administration<br>Code 012A<br>Naval Postgraduate School<br>Monterey, CA 93940 | 1 |

~~U2022~~